Name (print neatly): __________________________

Problems 1-26 are 3 points each, for 78 points
Problems 27-29 are 7 points each, for 21 points

99 points + 1 bonus point = 100 total.

This test is worth 20% of your final grade. You must put your answers on the answer form using a #2 pencil. Be sure to have your UIN written on your answer form under the space for code number, left justified, no spaces. You may assume code is in C++ unless specified otherwise. This test is open book and open notes. You may assume that segments of code are part of a larger working program. You have two hours.

1. Consider the struct declaration and code shown at right below intended for setting and displaying employee information for a single employee. Which of the following best describes the correctness of this code segment?

a) The code will compile and run correctly as shown
b) The code will not compile and has one error
c) The code will not compile and has two errors
d) The code will not compile and has three or more errors

```
struct Employee {
    int id;
    float hourlyWage;
    char code;
};

Employee anEmployee;
Employee *pEmployee = &anEmployee;
anEmployee.id = 2547;
anEmployee.hourlyWage = 8.0;
(*pEmployee).code = 'D';

cout << pEmployee.id;
cout << *pEmployee.hourlyWage;
cout << anEmployee->code;
```

2. Consider a large array of Employees, using the struct declaration from the previous problem. According to our discussion in class, which of the following is the best implementation so that we can both access the employee information in order according to the id field, and in addition also access employee information in order according to the code field?

a) When accessing by id, ensure the array is first sorted by id. Do this similarly for code
b) Create duplicate copies of the array, where one is always kept sorted by id, and the other is always kept sorted by code
c) Keep the array sorted by id, and access it sequentially when searching by code
d) Have separate arrays of pointers for id and for code, with the pointers implementing ordering of what they point to
3. Consider the struct declaration (shown at right) used to implement a list which has a sentinel node, so there is always at least one node on the list.

Which of the following is the best answer regarding a function declaration used to prepend nodes at the beginning of the list?

- a) The function declaration could be: `void prepend(int value, Node *pHead)`
- b) The function declaration could be: `Node * prepend(int value, Node *pHead)`
- c) The function declaration could be: `void prepend(int value, Node * &pHead)`
- d) More than one of the above answers is correct

4. Which of the following is the best description of how long it will take to sort elements of an array using the bubble sort algorithm as described in class?

- a) It takes a constant amount of time
- b) It is proportional to the number of elements
- c) It depends on the original order of the elements
- d) It depends on both the number of elements and the initial order of the elements

5. Which of the following is the best description of how long it will take to sort elements of an array using the selection sort algorithm as described in class?

- a) It takes a constant amount of time
- b) It is proportional to the number of elements
- c) It depends on the original order of the elements
- d) It depends on both the number of elements and the initial order of the elements

6. Consider gathering the unique student id numbers in an array for a class of 100 students, where the id numbers are randomly distributed in the array. When searching for a particular id number, on average how many id numbers will need to be examined? Assume we are using some sort of search as described in class. Choose the closest number if the exact number is not shown.

- a) 1
- b) 10
- c) 20
- d) 50

7. As in the previous problem, again consider gathering the unique student id numbers in an array for a class of 100 students, however this time the id numbers are carefully organized in the array. When searching for a particular id number, on average how many id numbers will need to be examined? Assume we are using some sort of search as described in class. Choose the closest number if the exact number is not shown.

- a) 1
- b) 10
- c) 20
- d) 50
8. Consider the function shown at right below. What value is displayed by the statement:
   \texttt{printf("\%d", fact1( 135));}
   \begin{verbatim}
   int fact1( int n)
   {   if (n < 10)
        return n;
    else
        return fact1( n / 10);  
   }
   \end{verbatim}
   a) 1  
b) 13  
c) 35  
d) 5

9. Consider the function shown at right below. What value is displayed by the statement:
   \texttt{printf("\%d", fact2( 5, 2));}
   \begin{verbatim}
   int fact2( int n, int x)
   {   if( n==0)
        return 1;
    else
        if( n>0)
            return fact2(n-1, x) * x;
        return 0;
   }
   \end{verbatim}
   a) 10  
b) 25  
c) 32  
d) 52

10. Consider the function shown at right below. For positive integers, how would you best describe the
    return value of calling function \texttt{fact3}?
    \begin{verbatim}
   double fact3( int x, int n)
   {   if( n.leq 0)  
        return 0;  
       return fact3( x, n-1) + x;  
    }
   \end{verbatim}
    a) $x$  
b) $x + n$  
c) $x \times n$  
d) $x^n$

11. Consider two programs that are written to implement finding the factorial of a number (e.g. factorial of
    3 is $3\times2\times1=6$). One program implements repetition using a \texttt{for} loop, and the second program
    implements repetition using \textit{recursion}. Which of the following is the best description of the relative
    amount of memory used by these two implementations?
    a) The two are the same
    b) The program using the \texttt{for} loop will use more memory because of the looping variable
    c) The recursive program will always use more memory because of the stack
    d) The recursive program will always use more memory for numbers larger than 2
    \begin{itemize}
        \item \textbf{also accepted:} c
    \end{itemize}
12. Consider the following declaration of a function intended to grow an array of integers by dynamically allocating memory:
   
   ```c
   void growArray( int oldSize, int newSize, int* pTheArray)
   ```

   What is the best description of whether or not this function declaration will work?

   a) The function will not compile correctly
   b) The function could compile correctly and run correctly
   c) The function could compile correctly, however the calling code will not have access to the new larger array.
   d) The function could compile correctly and grow the array correctly

13. Consider the following declaration of a function intended to grow an array of integers by dynamically allocating memory:
   
   ```c
   void growArray( int oldSize, int newSize, int** pTheArray)
   ```

   What is the best description of whether or not this function declaration will work?

   a) The function will not compile correctly
   b) The function could compile correctly and run correctly
   c) The function could compile correctly, however the calling code will not have access to the new larger array.
   d) The function could compile correctly and grow the array correctly

14. Consider the following declaration of a function intended to grow an array of integers by dynamically allocating memory:
   
   ```c
   void growArray( int oldSize, int newSize, int* &pTheArray)
   ```

   What is the best description of whether or not this function declaration will work?

   a) The function will not compile correctly
   b) The function could compile correctly and run correctly
   c) The function could compile correctly, however the calling code will not have access to the new larger array.
   d) The function could compile correctly and grow the array correctly

15. Consider reading in words from a file into an array called `allWords`, where you know ahead of time that there are always 1000 words, however for each word you need to allocate exactly the amount of storage for just the number of characters in that word. The declaration for this array will likely be:

   a) `char allWords[ 1000][ 1000];`
   b) `char allWords[ 1000][ n];`
   c) `char *allWords[ 1000];`
   d) `char **allWords;`

16. Consider reading in words from a file into an array called `allWords`, where you *don’t* know ahead of time how many words there are, and you need to allocate for each word exactly the amount of storage for just the number of characters in that word. The declaration for this array will likely be:

   a) `char allWords[ 1000][ 1000];`
   b) `char allWords[ 1000][ n];`
   c) `char *allWords[ 1000];`
   d) `char **allWords;`
17. Consider the following function declaration where a 2-d array has been passed:

```c
int countOnes( int numbers[ ][ 100])
```

In the above function declaration the value in the first set of braces may be left blank as shown, but the second dimension must be supplied. Why is this?

- **a)** Code in C can overwrite the end of an array, however for a 2-D array the formula used to find the ith row needs to know how many columns there are in each row.
- **b)** The size of the first dimension is always automatically supplied, even when not specified by the user, since it is included as part of the definition of every array.
- **c)** A NULL character is always inserted at the end of every row so that the compiler can tell where one row ends and the next begins, so the size of the first dimension is not necessary.
- **d)** The total size of the array divided by the size given in the second dimension is used internally to calculate the number for the first dimension, so the user need not supply it.

18. Consider the two approaches shown below to implement a **stack**:

<table>
<thead>
<tr>
<th>Approach A:</th>
<th>Approach B:</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagram](pTop pBottom)</td>
<td><img src="pTop" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Which is the best description of which of these two approaches is preferable?

- **a)** Approach A is preferable for implementing a stack
- **b)** Approach B is preferable for implementing a stack
- **c)** Neither one is preferable, they are both equally well suited for implementing a stack
- **d)** It depends on what kinds of stack operations will be implemented

19. Consider the two front-and-back pointer approaches shown below to implement a **queue**:

<table>
<thead>
<tr>
<th>Approach A:</th>
<th>Approach B:</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagram](pFront pBack)</td>
<td>![Diagram](pBack pFront)</td>
</tr>
</tbody>
</table>

Which is the best description of which of these two approaches is preferable?

- **a)** Approach A is preferable for implementing a queue
- **b)** Approach B is preferable for implementing a queue
- **c)** Neither one is preferable, they are both equally well suited for implementing a queue
- **d)** It depends on what kinds of queue operations will be implemented
20. Consider the function shown at right below used to traverse and display the values on a linked list. What is the best description of this function when called with pHead pointing to a non-empty list?

- **a)** It works correctly
- **b)** It traverses and displays the list, however it destroys the list head pointer in the process
- **c)** It does not compile
- **d)** It compiles but gives a run-time error

```cpp
void displayList(Node *pHead)
{
    while (pHead != NULL) {
        cout << pHead->data << " ";
pHead = pHead->pNext;
    }
cout << "\n\n";
}
```

21. Consider the struct declaration at left below that is used to implement a doubly-linked list. If there are multiple nodes on the list and we want to insert a new node before the current node in the middle of the list, which of the following is true about the insertNode(...) function?

```cpp
struct Node2 {
    Node2* pPrev;
    int data;
    Node2* pNext;
};
void insertNode(Node2 * &pCurrent, int input)
{
    Node2 *pTemp = new Node2;
pTemp->data = input;
pTemp->pPrev = pCurrent->pPrev;
pTemp->pNext = pCurrent;
pCurrent->pPrev = pTemp;
}
```

- **a)** It could be fixed by reordering the current instructions.
- **b)** To fix it would require modifying one instruction.
- **c)** To fix it would require adding an additional instruction.
- **d)** To fix it would require modifying or adding more than one instruction.

22. Consider the section of code shown at right below. What is displayed in the output?

- **a)** 2
- **b)** the address in memory of x
- **c)** the address in memory of xPtr
- **d)** the address in memory of zPtr

```cpp
int x, y, z;
int *xPtr, *yPtr, **zPtr;
x = 2; y = 7; z = 9;
xPtr = &x;
yPtr = &y;
zPtr = &xPtr;
cout << **zPtr;
```
Consider a maze program segment shown at right. It is similar, though not exactly the same, as what we discussed in class. Note that there are a total of 20 array values of zero. This function will be called as:

```cpp
makeMove( start);
```

23. What is the output from running this code?
   a) A solution path in order, displayed once.
   b) A solution path in reverse order, displayed once.
   c) A solution path in order, along with numbers of extra squares visited along the way.
   d) A solution path in reverse order, along with numbers of extra squares visited along the way.

24. How many unique array positions containing 0 are visited by this algorithm, including the start and goal squares?
   a) 10
   b) 11
   c) 14
   d) 19
   e) 20

25. If the values of `start` and `goal` are swapped, how many unique array positions containing 0 are visited by this algorithm, including the start and goal squares?
   a) 10
   b) 11
   c) 14
   d) 19
   e) 20

26. If the `start` and `goal` values remain as originally shown, however the `moves` array initial values are set to `{10, -1, 1, -10}`, how many unique array positions containing 0 are visited by this algorithm, including the start and goal squares?
   a) 10
   b) 11
   c) 14
   d) 19
   e) 20
27. (7 points) What is the output of the C++ code segment shown below when it is called with 
confuse4()? Assume the x and y variables declared at the top of the program segment are declared 
as global variables.

```cpp
int x=3, y=4;       // global variables

void confuse1(int y, int &x)
{
    x++;
    y++;
}

void confuse2(int *b, int x)
{
    (*b)++;
    y = x++;  
}

void confuse3(int &a, int *x)
{
    a = *x;
    (*x)++;
}

void confuse4()
{
    x=2;
    confuse1( x, y);
    confuse2( &x, y);
    confuse3( x, &y);
    cout << "x + y = " << x + y << endl;
}
```

a) x + y = 9  
b) x + y = 10  
c) x + y = 11  
d) x + y = 12

28. (7 points) Consider the segment of code shown at right below taken from the anagram program 
discussed in class. What is the second line of output when calling this with:

```cpp
char anagramString[ 81] = "abcde";
anagram( anagramString, 2);  
```

a) ab  
b) abc  
c) abcde  
d) abced  
e) abdce
29. (7 points) Carefully consider the C/C++ program segment given below, called with: do29()

```
struct Node {
    int data;
    Node *pNext;
};

void fcn29a( Node *pHead)
{
    while( pHead != NULL) {
        cout << pHead->data << " ";
        pHead = pHead->pNext;
    }
}

Node * fcn29( Node *pHead)
{
    Node *pTemp;
    if (pHead->pNext == NULL)  {
        return pHead;
    }
    else {
        pTemp = fcn29(pHead->pNext);
        pHead->pNext->pNext = pHead;
        return pTemp;
    }
}

void do29()
{
    int number = 0;
    Node *pHead = NULL;
    Node *pTemp;
    cout <<"Enter numbers, then -1: ";
    while ( number != -1) {
        cin >> number;
        if (number != -1) {
            pTemp = new Node;
            pTemp->data = number;
            pTemp->pNext = pHead;
            pHead = pTemp;
        }
    }
    pTemp = pHead;
    pHead = fcn29( pHead);
    fcn29a( pHead);
}
```

If the input is:

```
23 7 62 4 17 -1
```
then what is the output?

a) The reverse of the input, excluding -1
b) The same as the input, excluding -1
c) The same as the input, including -1
d) It does not compile correctly.
e) None of the above